



C# Avançado

ADO.NET Entity Framework




Tópicos Abordados

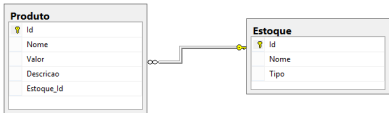


- Papel do ADO.NET Entity Framework
- Entities
- Camadas para o mapeamento
- Abordagens para o ORM
- A classe *DbContext*
- Manipulação de entidades
- Lazy loading e eager loading
- LINQ to entities
- Estados de entidades


Dois Universos Distintos



- Bancos de dados usam o modelo relacional, baseado em tabelas e colunas



- Objetos usam o modelo orientado a objetos, baseado em classes, fields, properties



ADO.NET Entity Framework

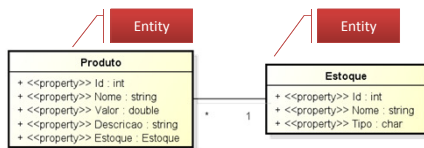


- A função do ADO.NET EF é juntar estes dois universos
 - Mapeamento objeto-relacional (ORM)
- O desenvolvedor trabalha apenas com o modelo de objetos
- Todas as operações no banco de dados ficam a cargo do ADO.NET EF

Entities



- Entities (entidades) são classes que representam conceitos da aplicação, que são mapeados para tabelas no banco de dados



O mapeamento pode ser totalmente customizado

Uma entidade não precisa ser necessariamente mapeada para apenas uma tabela

Definindo Entities



- Uma entidade é definida como um *POCO* (Plain Old CLR Object)
 - Não precisa estender de outra classe
 - Possui um construtor padrão
 - Contém properties

```
public class Produto
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public decimal Valor { get; set; }
    public string Descricao { get; set; }
    public int Estoque_Id { get; set; }
    public virtual Estoque Estoque { get; set; }
}
```

Properties que definem relacionamentos precisam ser definidos como *virtual*

Definindo Entities



```


public class Estoque
{
    public Estoque()
    {
        this.Produtos = new HashSet<Produto>();
    }

    public int Id { get; set; }
    public string Nome { get; set; }
    public string Tipo { get; set; }
    public virtual ICollection<Produto> Produtos { get; set; }
}

```

Relacionamento com
vários produtos

Camadas para o Mapeamento



- A criação do mapeamento exige a definição de três camadas
 - Lógica
 - Definição do modelo relacional
 - Definida pela SSDL (Store Schema Definition Language)
 - Conceitual
 - Definição das entidades
 - Definida pelo CSDL (Conceptual Schema Definition Language)
 - Mapeamento
 - Mapeamento entre o modelo relacional e entidades
 - Definido pelo MSL (Mapping Specification Language)

SSDL, CSDL e MSL



- O desenvolvedor não precisa se preocupar com estes arquivos
 - Apenas o arquivo *.edmx importa



Abordagens para o ORM

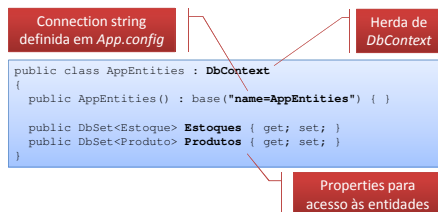


- O ADO.NET suporta três abordagens para gerar o mapeamento objeto-relacional
 - Model First
 - As entidades são criadas por primeiro
 - O banco de dados é criado com base nas entidades
 - Database First
 - O banco de dados é criado por primeiro
 - As entidades são geradas com base nas tabelas e colunas existentes no banco de dados
 - Code First
 - As entidades são criadas por primeiro
 - Não existe a necessidade de configurar o mapeamento usando CSDL, SSDL e MSL

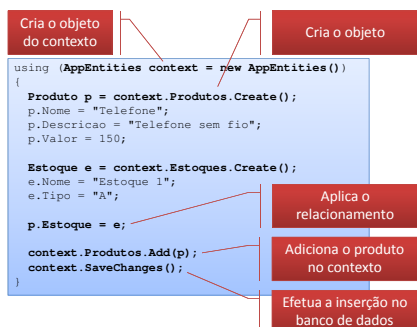
A Classe *DbContext*




- Um objeto desta classe é o ponto de entrada para o uso do ADO.NET EF



Inserindo Registros



Alterando Registros



```
using (AppEntities context = new AppEntities())
{
    Produto p = context.Produtos.Find(2);
    p.Descricao = "TV de LED";
    context.SaveChanges();
}
```

Obtém a entidade com base na sua chave

Atualiza a entidade no banco de dados

Excluindo Registros



```
using (AppEntities context = new AppEntities())
{
    Produto p = context.Produtos.Find(3);
    context.Produtos.Remove(p);
    context.SaveChanges();
}
```

Obtém a entidade com base na sua chave

Remove a entidade do contexto

Exclui o registro do banco de dados

Listando Registros



```
using (AppEntities context = new AppEntities())
{
    foreach (var p in context.Produtos)
    {
        Console.WriteLine(p.Nome);
    }
}
```

Entidades cadastradas no banco de dados

Lazy Loading e Eager Loading



- Quando existem relacionamentos entre entidades e estes relacionamentos precisam ser lidos, é possível utilizar
 - Lazy loading
 - A leitura dos dados do relacionamento é feita apenas no momento do acesso à entidade
 - Eager loading
 - A leitura dos dados do relacionamento é feita no momento que a "entidade-pai" é carregada
- O lazy loading é usado por padrão, mas isto pode ser alterado

```
context.Configuration.LazyLoadingEnabled = false;
```

Lazy Loading



```
using (AppEntities context = new AppEntities())  
{  
    foreach (var p in context.Produtos)  
    {  
        string nome = p.Estoque.Nome;  
    }  
}
```

Os dados do estoque são carregados no momento que a entidade é acessada

É realizada uma query para buscar os produtos e, para cada estoque, são executadas outras queries

Eager Loading



Carrega o relacionamento *Estoque* ao carregar os produtos

```
using (AppEntities context = new AppEntities())  
{  
    foreach (var p in context.Produtos.Include("Estoque"))  
    {  
        string nome = p.Estoque.Nome;  
    }  
}
```

É realizada apenas uma query, que faz um join entre produtos e estoques e carrega todas as entidades

Lazy Loading ou Eager Loading?



- Depende de cada caso
- Considerar as vantagens e desvantagens de cada abordagem
- O lazy loading só carrega dados quando eles forem usados, mas pode aumentar o número de queries executadas no banco de dados
- O eager loading diminui o número de queries executadas no banco de dados, mas pode trazer muita informação desnecessária

LINQ to Entities



- A linguagem LINQ pode ser utilizada para obter informações do banco de dados

```
using (AppEntities context = new AppEntities())
{
    var q = from p in context.Produtos
            where p.Estoque.Nome.EndsWith("A")
            select p;

    foreach (var p in q)
    {
        string nome = p.Nome;
    }
}
```

Produtos que estão em estoques
cujo nome termina com "A"

Estados de Entidades



- O objeto *DbContext* é capaz de gerenciar entidades
- Entidades que estão sob o controle do *DbContext* estão attachadas
- Apenas entidades attachadas são sincronizadas com o banco de dados
- Estados
 - *Added*: adicionada
 - *Deleted*: excluída
 - *Modified*: modificada
 - *Unchanged*: não modificada
 - *Detached*: não attachada ao contexto

Cenário 1

Softblue

- Criação de uma nova entidade

```

Produto p = context.Produtos.Create();
...
context.Produtos.Add(p);
context.SaveChanges();

```

Cenário 2

Softblue

- Carregamento e alteração de uma entidade

```

Produto p = context.Produtos.Find(1);
p.Nome = "Notebook";
context.SaveChanges();

```

Qualquer operação de carregamento traz os dados para o contexto

No caso de exclusão, funciona da mesma forma. A diferença é que o estado muda para *Deleted*

Cenário 3

Softblue

- Carregamento e desattachamento de entidade

```

Produto p = context.Produtos.Find(1);
context.Entry(p).State = EntityState.Detached;

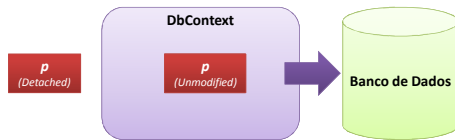
```

Alterações feitas na entidade não vão mais ser sincronizadas com o banco de dados na chamada *SaveChanges()*

Cenário 4



- Atachamento de entidade existente e modificada fora do contexto



```
Produto p = ...  
context.Entry(p).State = EntityState.Modified;  
context.SaveChanges();
```